

INTRODUCTION TO POS: A PROTOCOL OPERATIONAL SEMANTICS

JEAN-LUC KONING

LEIBNIZ-ESISAR, 50 rue de Laffemas - BP 54
26902 Valence cedex 9, France

PIERRE-YVES OUDEYER

Sony CSL Paris, 6 rue Amyot
75005 Paris, France

Received (to be inserted
Revised by Publisher)

In this paper we propose a system for representing interaction protocols called POS which is both Turing complete and determine a complete semantics of protocols. This work is inspired by the Structured Operational Semantics in programming languages. We precisely define POS and illustrate its power on an extended example.

Keywords: interaction protocols, operational semantics, inter-agent communication, collaboration, information agents.

1. Introduction

Multiagent systems are based on the idea that by gathering simple autonomous systems (agents) in a society and endowing them with interaction capabilities, such a society can display complex behaviors (compared to each agent's capabilities) that are not encoded in any individual entity in particular. The society's intelligence is distributed among its components.

A fundamental characteristic of these systems is their interactions' ability. The work presented in this article should be seen in the context of an interaction oriented-programming philosophy which argues that the power of these systems dwells less in the agents' intelligence than in their interactions. Our position is very close to the general approach described in ¹ that orchestrates the interactions among agents in settings such as open information environments.

Originally, approaches in the design of multiagent systems required full details of the internal structure of agents prior to designing their communication patterns. This led to hardwired interactions among agents that prevent them from being fully used in open information environments. More recent works (such as ²) argue for the protocols to be specified separately from the point of view of the agents. Besides, a flexible declaration mechanism for specifying interactions seems to be required.

We agree with this view and aim at making interactions as efficient as possible. Interactions should be powerful and expressive—being a Turing complete calculus system would be welcome—, and “light”, i.e., of a low algorithmic complexity.

This paper is organized as follows. We start with a discussion about standard ways interaction protocols have usually been tackled and show some of their intrinsic limits. We then propose a formalism for describing the interaction protocols semantics (POS). An example to illustrate the power of the formalism is also given as well as implementation details. We conclude with some extensions and evaluation of the merits of our approach.

2. Traditional Descriptions of Interaction Protocols

2.1. Example of a Learning Protocol

In this paper we are going to look at a general yet multi-faceted protocol. We could have taken the well known Contract Net protocol ³ but it seemed to us that Sian’s learning protocol ⁴ was showing a greater complexity (its graph involves a greater number of conversational states) and could thus better exemplify the expressive power to the operational semantics we are developing in this paper.

Let us first describe this protocol in natural language. Each agent stores its experience database (*eb*) and has a knowledge base (*kb*). When an agent elaborates a hypothesis or a new proposal from its experience base it puts it forward to the other agents. Those agents can either answer

- by confirming the hypothesis if they acknowledge its truth when looking up in their *eb* or *kb*.
- by modifying it if they can provide a more general or specific proposal from their *kb*.
- by signaling their disagreement if there is an opposite proposal in their *kb*.
- by signaling they do not have any opinion in case they do not have anything in their *eb* or *kb* that permits them to confirm, generalize, specify or counter such hypothesis.

The agent which elaborates the initial hypothesis collects each agents’ answer and evaluates its truth according to the answers received. Then, given a criterion, the hypothesis is considered as agreed or is abandoned. In the former case, the agent signals the agreement to the others which then accept it and store it in their *kb*. In the latter case, the hypothesis is not stored in the agents’ *kb*. Apart from this collective learning process, any agent may state a proposal to the others when it already belongs to the agent’s *kb*. In this case the other agents readily accept the hypothesis and update their *kb*.

One notices that a natural language description of a protocol gives a good idea of what will take place but leaves room to numerous imprecisions and interpretations,

which obviously is unacceptable in a data processing context such as ours. The goal is thus to express this protocol in a non-ambiguous formalism constituting a complete description of the protocol.

Often the formalization of such a protocol is rendered by means of finite automata each agent must possess. Figure 1 gives the one for Sian's protocol. A possible interpretation of a transition can be, "if the agent is in state *Decision* it can go to state *End* upon sending message *Withdraw*".

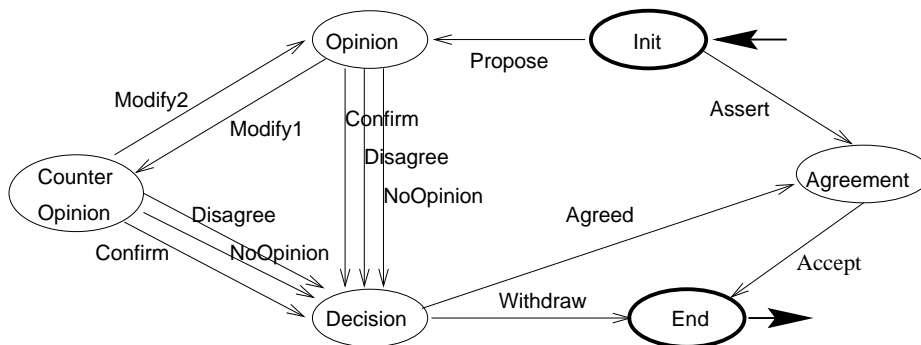


Figure 1: State transition graph of Sian's protocol.

2.2. Finite State Automata

When agents discuss by means of an interaction protocol, they each own a representation of this protocol which allow them to know the messages they can receive, the messages they can send, and at what time. This enables them to know where the conversation is at.

Finite state automata is a largely used formalism for describing interaction protocols. Each agent possesses a finite state automaton whose states represent the current state of the conversation they are carrying out, and the transitions correspond to the messages exchanged. There exists one or more initial and final conversation states. This approach is seen in lots of papers, and have sometimes been defined in a ad hoc manner in order to comply with applications' necessities. See ⁵, ⁶, etc., to name a few. ⁷ also acknowledges the shortcoming of using finite state machines with an associated textual description because this does not adequately captures constraints that cross different protocol levels. They provide a complete formal specification of protocols in the Z specification language which helps go all the way to implementation and simulation. That approach relies on strongly typed message parameters like ours.

Let us point out some limits of a finite state automaton description such as the one given in the previous section.

1. First of all, such a specification is far from being complete. When an agent communicates it sends and receives messages. It is important for this agent to know that a transition going out of a given state either corresponds to the sending or the reception of a message. However no distinction is usually made in the above description of the protocol. As a consequence one has to interpret this specification. Most of the time, one considers the messages are alternately sent and received, i.e., if an agent's former transition happened with a received message, then the next one will be a message sending, and vice versa.

Furthermore, not distinguishing between the sending or the reception of messages implies there is no distinction between the agent initiating the conversation and the other agents. During the course of the conversation not all agents have the same roles but this issue is eluded here. This shows a formal imperfection. However, this can be quite easily corrected by a protocols operational semantics as we shall see in section 3.

2. Again, as far as the specification's accuracy and autonomy, let us note that this formalism only describes the possible series of messages. It is only a syntactic specification whereas its semantics aspect conveyed in the text (like "storing a piece of information in the knowledge base *kb*") is forgotten. One still needs the textual description to use/implement the protocol.
3. Then, from the perspective of expressing protocols, finite automata are in fact not very practical. The inherent complexity of the protocols' representation with such a formalism is huge. For example, one must mention that the specification of Sian's protocol given above is only valid for two agents. Indeed, the graph only provides one occurrence of messages such as Confirm, Disagree and NoOpinion. Besides, not only does it turn out that trying to adapt such representation for more than two agents is not immediate, but this quickly leads to a combinatoric explosion. Furthermore, this will require a different automaton for each number of agents using Sian's protocol.
4. More fundamentally, finite state automata constitute a very poor calculus system. They are only capable of accepting regular expressions. However, one of the interaction protocol's objectives is to be powerful. Using this type of formalism is thus basically not adapted. For instance, it is not possible to describe a protocol ending in an *end* state after any message series of type $msg_1^n msg_2^n$ and only these ones. Moreover, it is also impossible to describe valid protocols for n agents ($\forall n \geq 1$). Therefore, protocols for open multiagent systems (i.e., with a variable number of agents along the time) are out of the finite automata's scope.

However, a number of articles such as ^{8,9} have considered finite automata because they are simple objects which are well mastered at the theoretical level.

Lots of their properties are known, and that is very interesting when dealing with the interaction protocols validation. The difficulty to convey slightly complex protocols has sometimes lead to abandon them for more compact and expressive formalisms such as Petri nets (see ¹⁰ for example).

2.3. Representing Protocols with Petri Nets

Petri nets are a good compromise between power of expression and analysis capabilities. Indeed, they allow to synchronize automata running in parallel. Moreover, they are a more compact representation than standard finite state automata. Lots of theoretical results have been found on Petri nets, they are thus interesting from a validation point of view. Let us mention ¹¹ which makes use of them in order to formalize and study interaction protocols in multiagent systems. It is also possible to formally describe negotiation protocols with n agents as well as validate several properties.

However, Petri nets are quickly limited. This is still only a syntactic protocol specification. One can thus only validate syntactic properties such as “the number of states of a negotiation process is finite” or “all the protocol’s messages are used”. Furthermore, representing interaction protocols with Petri nets always end up in a combinatoric explosion when the number of agents increases. And if some properties can theoretically be verified, this becomes intractable due to the inherent algorithmic complexity.

A Petri net representation of a protocol can quickly become very large and thus difficult to deal with. For example, Sian’s protocol for two agents leads to a Petri net with 26 places and 34 transitions and half of the places and transitions need to be added whenever a new agent is participating in the interaction.

This paper introduces a new formalism for describing interaction protocols and solves the problems highlighted above.

3. The POS Formalism: Definitions

In the remaining of this paper, each agent is assumed to encompass a FIFO mailbox and an identification number since all agents need a means to identify the other agents. We define here an interaction protocol as a production system owned by each agent.

Definition 1. An **interaction protocol** is a set of production rules of one of three types:

type 1

$$\underbrace{\langle \langle \overbrace{(Skeleton, Parameter)}^{\text{parametrized state}}, \phi(world) \rangle \rangle}_{\text{premise}} \xrightarrow{[Send]} \underbrace{\langle \langle (Skeleton', Parameter'), [A(world)] \rangle \rangle}_{\text{conclusion}}$$

type 2 $\langle \langle (Skeleton, Parameter), \phi(world) \rangle \rangle \xrightarrow{msg} \langle \langle (Skeleton', Parameter'), [A(world)] \rangle \rangle$

type 3 $\langle\langle \text{Skeleton}, \text{Parameter} \rangle, \phi(\text{world})\rangle \xrightarrow{\varepsilon} \langle\langle \text{Skeleton}', \text{Parameter}' \rangle, [\mathcal{A}(\text{world})]\rangle$

The definition of the variables are the following.

Skeleton is a character string. The purpose of such a field is to help protocol designers design a protocol in a clear manner. There is no theoretical need for it because this could be included in the parameter field. But it appears to be very useful for building protocols incrementally.

Parameter is a tuple of abstract type pattern matchable objects (with such primitive types as `Int`, `Char`, `List`, `Tuple`) and non-pattern matchable objects but which can be designated (these may be application specific). Two types of variables may be handled: (1) pattern variables, such as u , v , and w in a 3-tuple (u, v, w) , or *Statep* in the soccer robot example (see section 6) ¹² which mean “anything”, and (2) object identifiers such as p like in *Propose(p)*.

An integer pattern, for example, can be defined in an ML notation* by

```
pattern Integer = Zero | Succ(n)
```

In parameter $(\text{Succ}(n), 2, \text{Succ}(\text{Succ}(\text{Zero})), p)$ the phrase $\text{Succ}(n)$ is a pattern of a non negative Integer, 2 is an `Int`, $\text{Succ}(\text{Succ}(\text{Zero}))$ is a totally defined `Integer`, and p is the name of a specific object of the application (which can be designated by p).

$\phi(\text{world})$ is a predicate on the world which can be evaluated by the agents using the protocol. Let us note that no assumption is made on the way each of the agents conducts this evaluation. For example, ϕ could either mean “*does the first component of the current parameter have the pattern $\text{Succ}(n)$?*” or “*is my energy level high enough?*”. These two types of evaluation may relate to conversational states or to the world external to the protocol.

msg is a message pattern whose format is identical to a parameterized state: $(\text{Skeleton}, \text{Parameter})$. For example, $(\text{HereIsYourCommand}, (\text{Succ}(n), \text{Succ}(m), p))$ can denote $(\text{HereIsYourCommand}, (33, 2, [[23] [45]]))$.

[Send] is a list of sendings, where a sending is one of the following actions:

- *SendToId id msg*: send message msg to agent id .
- *SendToGroup list msg*: send message msg to the agents listed in $list$.
- *SendToAll msg*: send message msg to all agents involved in the conversation (except for the sending agent of course).

[$\mathcal{A}(\text{world})$] is a list of $\mathcal{A}(\text{world})$ which are the agent’s actions/side effects/behaviors onto the world. This is a very general notion which may go from removing a piece of information from a knowledge base to seeking an energy source.

*See ¹³.

Furthermore, type 1 productions correspond to sending messages, type 2 productions correspond to reading a message, and type 3 productions are transitions without any message exchange. At least one parameterized state must be an initial state—there may be several—and there are zero or more final states conveying the end of a conversation, i.e., the end of a protocol use.

Property 1 (Use of a protocol). In order to use a protocol, an agent must possess a copy of the set of production rules which define the protocol. Moreover, it must be capable of evaluating all the $\phi(\text{world})$, performing all the $\mathcal{A}(\text{world})$, and has to understand those rules. In particular, an agent must be able to accomplish the pattern matching, i.e., it must embody a small interpreter. Once it is decided that an agent will undertake a conversation with some other agents using a certain protocol, it must refer to the set of production rules and determine the possible transitions. That is:

- type 1 transitions are the ones whose parameterized state matches the current state and whose $\phi(\text{world})$ is verified.
- type 2 transitions respect the same conditions except that the corresponding message pattern must be first in the mailbox.
- type 3 transitions respect the same conditions as type 1 transitions.

It is obvious, and this is intended this way, that several productions will be possible for a given premise (part of a production before the \rightarrow connector). To cope with this a first alternative would be to leave the agent the freedom to choose which ever premise it wants. This choice could be based upon some weighing of the various rules. Another way to tackle this issue is by entering a conflict-resolution procedure such as choosing randomly one of the possible transitions.

Once a transition has been chosen, it is automatically fired by the agent. Any message sending attached to a production rule must be accomplished and all the $\mathcal{A}(\text{world})$ must be started before the next firing round starts. Once a message has been read and used, it is removed from the agent's mailbox, i.e., the corresponding transition has been fired.

Finally, when the agents want to start a conversation they each must possess a set of production rules—i.e., a protocol—but it does not have to be the same for every one. This set of productions will allow them to correctly interpret the messages they receive, take up the adequate behavior or perform the adequate side-effect, and send relevant messages for the ongoing conversation.

Let us formally define what a conversation is.

Definition 2. A **conversation** (or a discussion) among agents consists for each agent in using a protocol in order to understand the messages they receive, and send relevant messages. In other words, it is enough for the protocols to be compatible (i.e., any message sent (or received) in an agent's protocol can be dealt with as a received (or sent) message in the other agent's protocol).

Definition 3. A **skeleton automaton** of a protocol is the following finite automaton:

- the *states* are the skeletons of the parameterized states present in the protocols' rules.
- the *initial states* are the ones corresponding to the initial parameterized states of the protocol.
- the *final states* are the ones corresponding to the final parameterized states of the protocol if any.
- the production rules correspond to transitions between partner parameterized states. These productions are
 - *typed*, i.e., the transitions take place upon sending a message, receiving a message, or upon an ε -transition.
 - *labeled* by the message skeleton(s) of the corresponding transition.

Section 5 shows an extended example of such a skeleton automaton.

In no way are skeleton automata protocol specifications because a specification evade a great deal of information. However, they are both useful for getting an overall view of the protocol and for deriving some coarse grain properties.

Definition 4. A parameterized state S is **reachable** if there exists a series of transitions starting from an initial parameterized state and leading to a conclusion whose parameterized state is S

Definition 5. An interaction protocol is **terminable** if it owns at least one final state and one of them is reachable.

We shall see in section 5.2 how interesting non-terminable protocols can be.

Definition 6 (Ambiguity). A protocol is **ambiguous** whenever a sending and a reception rule conflict during the course of a conversation.

In short, a protocol is ambiguous when given a parameterized state the agent using the protocol can get to another state either through the sending or reception of messages.

This type of situation may enforce the power of a protocol but may also pose enormous problems. For instance, it may turn out that an agent always chooses production rules with a sending action and thus never read its incoming messages. This might result in blocking some other agents waiting for its answer and thus might lead to an overall system inefficiency. In order to remedy this problem, one introduces an over-specification on ambiguous protocols by aiming at favoring the productions related to the reading of a message compared to productions related to the sending of a message. However, this problem disappears when one imposes a conflict resolution procedure, as described hereafter, based on a random choice of a rule.

Definition 7. An ambiguous protocol is **altruistic** if its usage is over-specified in the following way: if at one point production rules with the sending or the receiving of messages is possible then an agent must choose a production rule with a reading of messages.

With respect to this, it is good to define a default conflict resolution policy. Presumably, the best policy consists in randomly choosing among possible rules. Granting an agent a specific conflict resolution strategy that is external to the protocol means the protocol is not self-sufficient in that its entire semantics is not totally included in its description.

The choosing among the rules is not given by a protocol but still takes on critical importance. On some occasions, it is good to trade this default arbitration rule (random choice) for an altruistic policy, i.e., a sole preference for reception rules. Nevertheless such strategies must be the more syntactical and the least semantic.

Both policies given above are totally syntactic.

4. Theoretical Considerations

4.1. Power of POS

Theorem 1. The calculus system defined by a set of POS production systems is Turing complete.

Proof Let us only look at production rules of type:

$$\langle (Skeleton, (l_1, l_2)), true \rangle \xrightarrow{msg} \langle (Skeleton', (l'_1, l'_2)), NothingToDo \rangle$$

where l_i are lists, i.e., symbol stacks. Productions of this type allow for the description of all two-stack finite automata, where the stacks are l_1 and l_2 , the states are the skeletons, and transitions are fired upon consumption of messages from the mailbox. In the usual setting of a two-stack finite automaton the received messages make up the alphabet of the input strings, where a string designates the series of messages in the mailbox. It is widely known that two-stack finite automata are Turing complete calculus systems¹⁴.

Since the productions just seen here are a subset of POS rules, one deduces that POS is Turing complete. \square

This property shows in a theoretical manner POS' superiority compared to the approach presented in section 2. Unlike finite state automata or even Petri nets POS's power does not hold to the detriment of its ease of use. The opposite is taking place. This paper exemplifies this claim on an extension of Sian's protocol (see section 5) as well as on real applications dedicated to autonomous robots (see section 6).

4.2. Semantics

POS endows also the advantage of giving a complete protocol semantics. The sets of production rules are sufficient by themselves (if necessary one can add the altruistic characteristic). Furthermore, it is an actual semantics in the sense of programming languages. As mentioned earlier POS has been inspired by the Structured Operational Semantics (SOS) ¹⁵. This has already been used in multiagent systems but rather on agents than interactions. For example, ¹⁶ makes use of the structured operational semantics for defining an abstract agent programming language. ¹⁷ also follows the structural operational semantics approach in modeling the operational semantics of their actor algebra by means of labeled transitions systems.

SOS enables to set up the semantics ¹⁸ of a language L via a formal description of an interpreter I of this language whose behavior is specified by the production rules. I is then an entity taking as argument a program P written in a language L , and an environment σ . I describes the behavior of $\langle P, \sigma \rangle$ which is a finite or infinite series like $\langle P, \sigma \rangle \rightarrow \langle P_2, \sigma_2 \rangle \rightarrow \langle P_3, \sigma_3 \rangle \rightarrow \dots$. If P stops the result is then $\langle END, \sigma_n \rangle$

When transposing this to our approach on protocols the set of possible messages is seen as an alphabet A . A language L' can then be defined as the following $\{m \in L' : \varepsilon | a \in A | m_1 m_2 / m_1 e t m_2 \in L'\}$ where strings in L' are message sequences. Then the set of productions defining a protocol can be seen as the definition of an interpreter, the series of received messages can be seen as a program to be interpreted, the world can be seen as the environment σ in SOS, the messages sent and the \mathcal{A} (world) can be seen as the side effects on σ , like a variable assignment in SOS for instance.

To keep on drawing a parallel with SOS, one can clearly see an analogy between languages with a non-deterministic semantics—those encompassing “random” instructions for instance—and interaction protocols since several transitions are often possible from a unique parameterized state.

Let us also note that during the “interpretation” of the program (made up of a series of messages) by the protocol, there is no way to know the final program in advance. Indeed, one does not know the series of messages that is to be received. By the way, this shows that the SOS philosophy is best suited for describing the semantics of this kind of program. In fact, one of its most salient features is that this goes from past to future (characteristic of a direct semantics) and that the meaning of the program is built as the text of the program is traversed (feature of interpreters). On the contrary, philosophies like the semantics by continuations seems definitely inappropriate since they go back from future to past and build the meaning of a program at once prior to its execution (feature of compilers).

Finally, specifying a protocol in POS amounts to defining a language interpreter. Such specification provides a semantics to the series of messages received. Not only is this very interesting as such, but also as far as the communication issue between agents in multiagent systems in general. As a matter of fact, POS defines as many languages as protocols. When one considers a protocol is used for communicating and understanding each other in a particular context, POS enables the creation of a

language for each context, where the term “language” is first taken in its computer science definition but also in a broader sense. This obviously was the prime goal for the creation of protocols, and this is what emerges a posteriori, which is a good indication this approach is sound.

To go even farther, let us note that although POS can be viewed as a calculus system, above all it is a tool for defining languages, as just shown. Languages define themselves calculus systems, and it seems that POS enables to define languages which are themselves Turing complete. Due to a lack of space, we will not be providing any proof of this in this article, but we have managed to write a small interpreter of a Turing complete imperative language with POS. One of the consequences of this is that one can now express a lot of things. For example, those languages can be used to define other languages/protocols via the definition of new interpreters in the image of writing an ML interpreter in ML. In particular, one can think of bootstrapping our primary language (i.e., initial protocol) by re-conveying its interpreter by means of itself!

4.3. Distributed Nature of Protocols

Let us go back to the general philosophy of this formalism. In fact, its basic principle can be stated as “what an agent has to do is somehow dictated by the other agents”. Since nevertheless all the agents have some freedom, it would be more exact to say that “the other agents are the one limiting the room to maneuver of each agent”. This approach stems from the speech act theory¹⁹ which consists in considering speech acts as actions. For POS, each speech act (or message) is attached to a side effect which boils down to molding the receiver’s room to maneuver. This approach fully takes root in the distributed approach of multiagent systems.

In our analogy with programming languages, a conversation in fact consists in an automatic distributed generation of programs, i.e., a sequence of messages received by each agent. However, in our approach these programs constitute the encoding of the “behavioral direction” an agent must go; in other words, it is a summary of what it is to do. The intelligence lies in the interaction. This position is very close to what is advocated in¹. Here a conversation automatically generates each agents’ behavior in a distributed fashion. And it does this dynamically while taking into account the currently evolving environment. The theoretical power of such a system enables to solve any Turing computable problem²⁰.

4.4. Formalism’s Effectiveness

Several formalisms from various fields are suited for theoretical studies, but they often are very far from data processing realities and are very hard to implement when one has to transform them in order to code them. POS has the advantage of being readily implementable in strongly typed functional languages such as ML¹³ or Haskell²¹. Thanks to their functional syntax, their management of abstract types and their pattern matching capabilities, POS protocols can virtually be coded

word for word in those languages. Furthermore, coding a protocol in one of these languages may be considered a protocol specification as such. These are very expressive languages which have been created with a sound and totally formalized semantics. They are at the heart of research works in formal semantics of programming languages. They are widely used by people working on formal methods both for proving programs and also for proving the security of usual communication protocols. In the latter domain some research teams make use of protocol specifications written in ML ²². We have used this language too for our applications. Languages such as Java/Pizza ²³ which are used in the multiagent system community could also have been used. In any case, it is not difficult to implement an interpreter that does pattern matching this way with any programming language.

5. Application Example

Having considered theoretical issues, let us now see some examples. We are going to present two versions of Sian's protocol ⁴ in order to exhibit the power of our formalism: one involving two agents as discussed in section 2.1, and one involving n agents

5.1. Sian's Protocol for Two Agents

We are going to model the protocol described at the beginning of this paper (see section 2.1). Here an agent starting the conversation is not going to have the same set of production rules as the one answering it. It will have an isomorphic set. Figure 2 enables to easily follow the unfolding of the automaton corresponding to the set of production rules of the agent starting the conversation. Messages have been defined as parameterized states composed of skeletons. Since the protocol given here is very simple it has been more appropriate to use the message's patterns instead of their skeleton. This is also what is used for the protocol's rules here under.

This automaton is quite different from the one given in figure 1. The protocol presented here is non-ambiguous, which simplifies things a lot. The counterpart of this is that it is necessary to have two sets of productions, one for the agent starting the conversation and one for the agent answering it. We shall see though in section 5.2 that it is possible to build a unique set of production rules for Sian's protocol involving n agents.

Let us first focus on the set of rules of the agent starting the conversation. Action *NOP* denotes the empty action, i.e., not doing anything. $eb - p$ or $eb + p$ denotes the removing or adding of a piece of knowledge p to or from the knowledge base eb . In a concern of keeping things as clear as possible, we have replaced the [*sendToAll(msg)*] by *msg* when writing the following protocols. We indicate the messages that are not broadcasted.

$$\begin{aligned} \langle (Init, p/p \in eb), WantToPropose(p) \rangle &\xrightarrow{Propose(p)} \langle (Opinion, p), NOP \rangle \\ \langle (Init, p/p \in eb), WantToAssert(p) \rangle &\xrightarrow{Assert(p)} \langle (AgreementPropose, p), NOP \rangle \end{aligned}$$

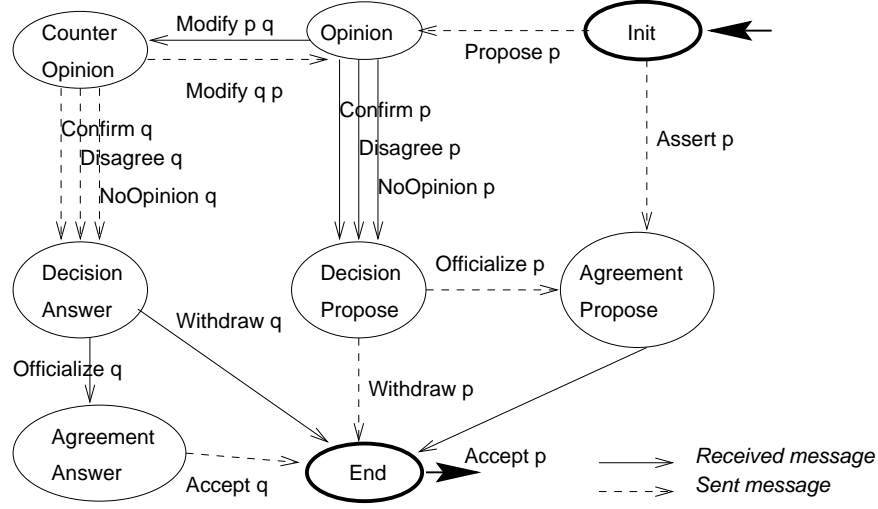


Figure 2: Automaton of Sian's protocol with two agents (for the agent initiating the conversation).

$$\begin{aligned}
 & \langle \langle \text{AgreementPropose}, p \rangle, \text{True} \rangle \xrightarrow{\text{Accept}(p)} \langle \langle \text{End}, p \rangle, (eb - p, kb + p) \rangle \\
 & \langle \langle \text{Opinion}, p \rangle, \text{True} \rangle \xrightarrow{\text{Confirm}(p)} \langle \langle \text{DecisionPropose}, \text{Confirm}(p) \rangle, \text{NOP} \rangle \\
 & \langle \langle \text{Opinion}, p \rangle, \text{True} \rangle \xrightarrow{\text{Disagree}(p)} \langle \langle \text{DecisionPropose}, \text{Disagree}(p) \rangle, \text{NOP} \rangle \\
 & \langle \langle \text{Opinion}, p \rangle, \text{True} \rangle \xrightarrow{\text{NoOpinion}(p)} \langle \langle \text{DecisionPropose}, \text{NoOpinion}(p) \rangle, \text{NOP} \rangle \\
 & \langle \langle \text{Opinion}, p \rangle, \text{True} \rangle \xrightarrow{\text{Modify}(p,q)} \langle \langle \text{CounterOpinion}, q \rangle, (eb - p, kb) \rangle \\
 & \langle \langle \text{CounterOpinion}, p \rangle, \text{WantToConfirm}(p), \text{True} \rangle \xrightarrow{\text{Confirm}(p)} \langle \langle \text{DecisionAnswer}, p \rangle, \text{NOP} \rangle \\
 & \langle \langle \text{CounterOpinion}, p \rangle, \text{WantToDisagree}(p), \text{True} \rangle \xrightarrow{\text{Disagree}(p)} \langle \langle \text{DecisionAnswer}, p \rangle, \text{NOP} \rangle \\
 & \langle \langle \text{CounterOpinion}, p \rangle, \text{HasNoOpinion}(p), \text{True} \rangle \xrightarrow{\text{NoOpinion}(p)} \langle \langle \text{DecisionAnswer}, p \rangle, \text{NOP} \rangle \\
 & \langle \langle \text{CounterOpinion}, p \rangle, q = \text{WantToModify}(p) \rangle \xrightarrow{\text{Modify}(p,q)} \langle \langle \text{Opinion}, q \rangle, (eb - p, kb) \rangle \\
 & \langle \langle \text{DecisionPropose}, s(p), \text{DecideToWithdraw}(s(p)) \rangle, \text{True} \rangle \xrightarrow{\text{Withdraw}(p)} \langle \langle \text{End}, p \rangle, (eb - p, kb) \rangle \\
 & \langle \langle \text{DecisionPropose}, s(p), \text{NotDecideToWithdraw}(s(p)) \rangle, \text{True} \rangle \xrightarrow{\text{Officialize}(p)} \langle \langle \text{AgreementPropose}, p \rangle, \text{NOP} \rangle \\
 & \langle \langle \text{DecisionAnswer}, p \rangle, \text{True} \rangle \xrightarrow{\text{Withdraw}(p)} \langle \langle \text{End}, p \rangle, (eb - p, kb) \rangle \\
 & \langle \langle \text{DecisionAnswer}, p \rangle, \text{True} \rangle \xrightarrow{\text{Officialize}(p)} \langle \langle \text{AgreementAnswer}, p \rangle, \text{NOP} \rangle \\
 & \langle \langle \text{AgreementAnswer}, p \rangle, \text{True} \rangle \xrightarrow{\text{Accept}(p)} \langle \langle \text{End}, p \rangle, (eb - p, kb + p) \rangle
 \end{aligned}$$

variable s denotes a state, where $s(p)$ is a pattern for $confirm(p)$, $Disagree(p)$ or $NoOpinion(p)$.

The set of production rules the second agent must possess is isomorphic to the one above.

$$\begin{aligned}
& \langle (Init, p/p \in eb), True \rangle \xrightarrow{Propose(p)} \langle (Opinion, p), NOP \rangle \\
& \langle (Init, p/p \in eb), True \rangle \xrightarrow{Assert(p)} \langle (AgreementPropose, p), NOP \rangle \\
& \langle (AgreementPropose, p), True \rangle \xrightarrow{Accept(p)} \langle (End, p), (eb - p, kb + p) \rangle \\
& \langle (Opinion, p), WantToConfirm(p) \rangle \xrightarrow{Confirm(p)} \langle (DecisionPropose, NOP) \rangle \\
& \langle (Opinion, p), WantToDisagree(p) \rangle \xrightarrow{Disagree(p)} \langle (DecisionPropose, NOP) \rangle \\
& \langle (Opinion, p), NoOpinion(p) \rangle \xrightarrow{NoOpinion(p)} \langle (DecisionPropose, NOP) \rangle \\
& \langle (Opinion, p), q = WantToModify(p) \rangle \xrightarrow{Modify(p,q)} \langle (CounterOpinion, q), (eb - p, kb) \rangle \\
& \langle (CounterOpinion, p), True \rangle \xrightarrow{Confirm(p)} \langle (DecisionAnswer, Confirm(p)), NOP \rangle \\
& \langle (CounterOpinion, p), True \rangle \xrightarrow{Disagree(p)} \langle (DecisionAnswer, Disagree(p)), NOP \rangle \\
& \langle (CounterOpinion, p), True \rangle \xrightarrow{NoOpinion(p)} \langle (DecisionAnswer, NoOpinion(p)), NOP \rangle \\
& \langle (CounterOpinion, True) \rangle \xrightarrow{Modify(p,q)} \langle (Opinion, q), (eb - p, kb) \rangle \\
& \langle (DecisionPropose, True) \rangle \xrightarrow{Withdraw(p)} \langle (End, p), (eb - p, kb) \rangle \\
& \langle (DecisionPropose, True) \rangle \xrightarrow{Officialize(p)} \langle (AgreementPropose, p), NOP \rangle \\
& \langle (DecisionAnswer, s(p)), DecideToWithdraw(s(p)) \rangle \xrightarrow{Withdraw(p)} \langle (End, p), (eb - p, kb) \rangle \\
& \langle (DecisionAnswer, s(p)), NotDecideToWithdraw(s(p)) \rangle \xrightarrow{Officialize(p)} \langle (AgreementAnswer, p), NOP \rangle \\
& \langle (AgreementAnswer, p), True \rangle \xrightarrow{Accept(p)} \langle (End, p), (eb - p, kb + p) \rangle
\end{aligned}$$

Let us now mention a few remarks. First of all, there is an advantage in having parameterized states whose parameter here is a single component which is a non pattern matchable object but which can be designated, i.e., a piece of knowledge p . This enables the agents to know the piece of knowledge they are discussing. This is necessary since the piece of knowledge may change during the course of the conversation in case a Modify message happens.

Secondly, even though the protocol's semantics is not very rich, it turns out its definition is fundamental. At various places, several choices are possible with respect to side effects. Here the side effects are the adding or removing of a piece of knowledge from the knowledge base. Each of the possible choices may lead to a different protocol semantics. For example, when one of the agents executes a $Modify(p,q)$, we have decided that both agents remove p from their knowledge base eb . This semantics choice permits to derive the following property.

Theorem 2. When two agents communicate using the two sets of productions given here above, if (1) they only put forward pieces of knowledge which do not belong to their own knowledge base, (2) both knowledge bases were finite at the beginning of the communication process, and (3) they are not altered by anything except the current protocol then the communication always comes to a halt.

Proof Either there is no **Modify** and then the ending is trivial, or there are some **Modify** and then the size of the experience base *eb* decreases by one unit each time. Since the number of pieces of knowledge must always be a non negative number, and since the agents can only execute a **Modify** when their experience base is not empty, the protocol always end up terminating. \square

One might object that this problem can as easily be dealt with using a finite state automata specification of the protocol. Of course the halting of a protocol is a syntactical property but here the proof itself relies on the protocols semantics. Had one not given a semantics specification to the protocol, it would have been impossible to prove this property.

Thirdly, this type of protocol makes very few assumptions on the internal structure of the agents. One just assumes they are capable of handling a protocol, they own one *eb* and one *kb*, they can add/remove pieces of knowledge to/from these knowledge bases, and they must be able to evaluate the predicates (they do so their own way). This highlights that protocols defined by POS have a relative independence with the agents' internal structure.

5.2. Sian's Protocol for n Agents

After giving a simple protocol example, let us now show the expression power of POS through the definition of a version of Sian's protocol for n agents, where each agent uses a same set of production rules. This implies to rewrite the protocol's textual description because the one given in section 2.1 was in fact not suited for the case were $n > 2$. From an agent's point of view, one is going to model the following.

- If an agent wants to put forward a piece of knowledge (the first agent to send a piece of knowledge is called the conversation initiator), it broadcasts it to all the other agents then waits for their answers. If it only receives **Confirm**, **Disagree**, **NoOpinion** messages, then it makes the decision of either officializing the acceptance of this piece of knowledge (**Officialize**), or officializing its reject (**Reject**). If it receives **Modify** messages, it chooses one and let it know to all the other agents (as well as the identity of the agent which sent the **Modify** message), then provides its viewpoint upon this piece of knowledge.
- If an agent is sent a proposition p , it sends its viewpoint then waits for either an **Officialize** message (in this case it sends back an **Accept** message), a **Withdraw** message, or a signal corresponding to a **Modify** (if it is a **Modify** message from another agent then it sends its viewpoint, if it is its own then it waits for the other's).

The set of production rules is given hereafter. It is called *SianN*. N denotes the number of agents participating in the conversation. id , $myid$ and $newid$ denote integers which designate the agents' identification. $myid$ denotes an agent's own identification. $[]$ denotes the empty list. $t :: q$ denotes the list obtained by concatenating t with list q . $N - 1$ is a shortcut to denote $Succ(Succ(\dots(Zero)))$

$$\begin{aligned}
& \langle \text{Init}, \text{True} \rangle \xrightarrow{\text{Propose}(p, myid)} \langle \langle \text{OpinionPropose}, (p, myid, N - 1, []) \rangle, \text{NOP} \rangle \\
& \langle \text{Init}, \text{True} \rangle \xrightarrow{\text{Propose}(p, id)} \langle \langle \text{OpinionAnswer}, (p, id) \rangle, \text{NOP} \rangle \\
& \langle \langle \text{OpinionPropose}, (p, myid, Succ(n), received) \rangle, \text{True} \rangle \xrightarrow{\text{Confirm}(p)} \\
& \quad \langle \langle \text{OpinionPropose}, (p, myid, n, (\text{Confirm}(p)) :: received) \rangle, \text{NOP} \rangle \\
& \langle \langle \text{OpinionPropose}, (p, myid, Succ(n), received) \rangle, \text{True} \rangle \xrightarrow{\text{Disagree}(p)} \\
& \quad \langle \langle \text{OpinionPropose}, (p, myid, n, (\text{Disagree}(p)) :: received) \rangle, \text{NOP} \rangle \\
& \langle \langle \text{OpinionPropose}, (p, myid, Succ(n), received) \rangle, \text{True} \rangle \xrightarrow{\text{NoOpinion}(p)} \\
& \quad \langle \langle \text{OpinionPropose}, (p, myid, n, (\text{NoOpinion}(p)) :: received) \rangle, \text{NOP} \rangle \\
& \langle \langle \text{OpinionPropose}, (p, myid, Succ(n), received) \rangle, \text{True} \rangle \xrightarrow{\text{Modify}(p, q, id)} \\
& \quad \langle \langle \text{OpinionPropose}, (p, myid, n, (\text{Modify}(p, q, id)) :: received) \rangle, \text{NOP} \rangle \\
& \langle \langle \text{OpinionPropose}, (p, myid, Zero, received) \rangle, \text{WantToOfficialize}(received) \rangle \\
& \quad \xrightarrow{\text{Officialize}(p)} \langle \langle \text{AgreementPropose}, (N - 1, p) \rangle, \text{NOP} \rangle \\
& \langle \langle \text{OpinionPropose}, (p, myid, Zero, received) \rangle, \text{WantToWithdraw}(received) \rangle \\
& \quad \xrightarrow{\text{Withdraw}(p)} \langle \text{End}, eb - p \rangle \\
& \langle \langle \text{AgreementPropose}, (Succ(n), p) \rangle, \text{True} \rangle \xrightarrow{\text{Accept}(p)} \\
& \quad \langle \langle \text{AgreementPropose}, (n, p) \rangle, \text{NOP} \rangle \\
& \langle \langle \text{AgreementPropose}, (Zero, p) \rangle, \text{True} \rangle \xrightarrow{\epsilon} \langle \text{End}, [eb - p, kb + p] \rangle \\
& \langle \langle \text{OpinionPropose}, (p, myid, Zero, received) \rangle, \text{Modify}(p, q, id) \in received \rangle \\
& \quad \xrightarrow{\text{Signal}(p, q, id)} \langle \langle \text{OpinionAnswer}, (q, id) \rangle, \text{NOP} \rangle \\
& \langle \langle \text{OpinionAnswer}, (p, id) \rangle, \text{WantToConfirm}(p) \rangle \xrightarrow{\text{sendToid}(\text{Confirm}(p)) \ id} \\
& \quad \langle \langle \text{WaitAnswer}, (p, id) \rangle, \text{NOP} \rangle \\
& \langle \langle \text{OpinionAnswer}, (p, id) \rangle, \text{WantToDisagree}(p) \rangle \xrightarrow{\text{sendToid}(\text{Disagree}(p)) \ id} \\
& \quad \langle \langle \text{WaitAnswer}, (p, id) \rangle, \text{NOP} \rangle \\
& \langle \langle \text{OpinionAnswer}, (p, id) \rangle, \text{HasNoOpinion}(p) \rangle \xrightarrow{\text{sendToid}(\text{NoOpinion}(p)) \ id} \\
& \quad \langle \langle \text{WaitAnswer}, (p, id) \rangle, \text{NOP} \rangle \\
& \langle \langle \text{OpinionAnswer}, (p, id) \rangle, q = \text{WantToModify}(p) \rangle \xrightarrow{\text{sendToid}(\text{Modify}(p, q, myid)) \ id} \\
& \quad \langle \langle \text{WaitAnswer}, (p, id) \rangle, \text{NOP} \rangle \\
& \langle \langle \text{WaitAnswer}, (p, id) \rangle, \text{True} \rangle \xrightarrow{\text{Officialize}(p)} \langle \langle \text{AgreementAnswer}, (p, id) \rangle, \text{NOP} \rangle \\
& \langle \langle \text{AgreementAnswer}, (p, id) \rangle, \text{True} \rangle \xrightarrow{\text{sendToid}(\text{Accept}(p)) \ id} \\
& \quad \langle \langle \text{End}, (p, id) \rangle, [eb - p, eb + p] \rangle \\
& \langle \langle \text{WaitAnswer}, (p, id) \rangle, \text{True} \rangle \xrightarrow{\text{Withdraw}(p)} \langle \langle \text{End}, (p, id) \rangle, eb - p \rangle
\end{aligned}$$

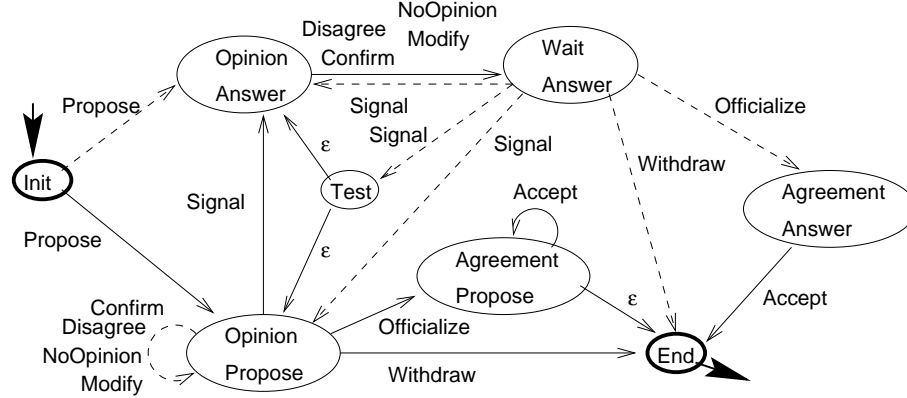


Figure 3: Automaton for SianN.

$$\begin{aligned}
 & \langle (WaitAnswer, (p, id)), True \rangle \xrightarrow{Signal(p, q, newid)} \langle (Test, (p, id, q, newid)), NOP \rangle \\
 & \langle (Test, (p, id, q, newid)), newid = id \rangle \xrightarrow{\epsilon} \\
 & \langle (Test, (p, id, q, newid)), newid < > id \rangle \xrightarrow{\epsilon} \langle (OpinionPropose, (q, id, N - 1, [])), NOP \rangle \\
 & \langle (Test, (p, id, q, newid)), newid < > id \rangle \xrightarrow{\epsilon} \langle (OpinionAnswer, (q, newid)), NOP \rangle
 \end{aligned}$$

The automaton is given on figure 3.

Let us draw some remarks. First, one can verify this set of production rules is valid for any n agents by instantiating N to the right n and by testing the protocol. The enormous advantage is that the protocol's representation does not grow when n increases. In fact, when some agents want to communicate via this set of productions, they have to know how numerous they are (n) and "build" the corresponding set of rules by assigning the right value for N , that we note $SianN[N := n]$.

Second, let us point out that the last three productions only serve the purpose of testing whether $newid = id$. But since there are three rules this is not an atomic operation. In case of an unlucky situation this could lead to the blocking of a protocol[†]. In order to solve this problem we have extended the way production rules could be constructed by adding a predicate upon received messages.

$$\langle (Skeleton, Parameter), \phi(world) \rangle \xrightarrow{msg / \Psi(msg)} \langle (Skeleton', Parameter'), [\mathcal{A}(world)] \rangle$$

For example, here, this enables to solve the problem of atomicity by replacing the three last productions by:

$$\langle (WaitAnswer, (p, id)), True \rangle \xrightarrow{Signal(p, q, newid) / (newid=id)} \langle (OpinionPropose, (q, id, N - 1, [])), NOP \rangle$$

[†]This is very close to synchronization issues in operating/distributed systems as described in ²⁴.

$$\langle (WaitAnswer, (p, id)), True \rangle \xrightarrow{Signal(p, q, newid) / (newid <> id)} \langle (OpinionAnswer, (q, newid)), NOP \rangle$$

During this conversation, an agent can either have the role of a proponent of a piece of knowledge or an evaluator of a piece of knowledge. Agents can switch roles during the course of a discussion upon incoming **Modify** messages.

6. Other Application Domains

Not only is POS suited for dealing with interaction protocols in multiagent systems as exemplified by Sian's protocol but also it is capable of addressing interaction protocols with physical autonomous agents.

In ²⁵ we deal with a world containing two types of entities : *objects* placed randomly in the world and unable to move by themselves, and *agents* capable of moving and whose initial location is chosen at random. The problem is to "handle" as many objects as possible in the most efficient and quick manner. The word "handle" is left unclarified in order to keep the protocol as general as possible. By the way, this handling could itself be taken care by another protocol to be connected with the one presented in ²⁵. On the other hand, in order for an object to get handled, it is first necessary to find it, and secondly S agents have to be involved in this handling, which means they have to be located exactly where the object is. Any agent is capable of participating in the handling of an object. However, protocols allow them to refuse helping in a handling whenever they do not want to.

In the protocol we have modeled (called **Collect**) each agent is looking for an object. Then, when it is finding one it tries to recruit four other agents to help it handle the object. If it does not manage to do so it moves away from the object and starts again seeking a new one. In case of success, it waits for the four recruited agents to arrive, they then handle the object and they all start over looking for an object by themselves. To avoid dispersion, whenever an agent has declared it would go help handle an object it becomes unavailable to the others. This makes it reply with a negative answer to later recruitment queries. It goes straight to the object location and sends a message to the calling agent that it has arrived.

This type of protocol definitely applies to a world where several kinds of objects exist and where no one agent is capable of handling all of them (an agent has limited capabilities). The consequence is that an agent asking for help, will be replied with a negative answer. Besides, the protocol can easily be transformed to take into account a variable number of agents, or even a particular set of physical agents (robots), in order to handle each object. For example, an agent may have to lift, another to dig, another to clean and another may have to open. The **Collect** protocol deals with a number of agents fixed at the beginning of the conversation. But one can imagine to adapt this protocol to an open system. When a new agent wants to enter a conversation, it sends a specific message to all agents involved in the

conversation and they modify their own version of the protocol accordingly. Such a modification is not mundane but still can be accomplished.

In the area of mobile computing more and more agents are used to fetch pieces of information on the internet. However, a single agent cannot be in two locations at the same time, whatever his intelligence. Its field of investigation is therefore very limited. On the contrary, a fleet of little and possibly heterogeneous agents may greatly improve their overall action when making use of such a protocol. For example, when an agent is finding a piece of information that is particularly interesting, it may call other agents to help evaluate/treat it, since they may not each have all the necessary capabilities to do the job. This processing of a piece of information may itself be accomplished by a dedicated protocol. This leads for example to composing the Collect protocol discussed so far with another more specialized one.

Another field of interest where such a protocol could be used is collective robotics. ²⁶ presents a situation where robots have to work and cooperate in a harbor. More generally, all the problems identified as being of the forage principle (inspired by insects societies) can be tackled by our protocol. In ¹² we have examined several basic behaviors of a JavaSoccer simulation environment and used it along with JavaBots ²⁷ to demonstrate POS' capabilities. We have modeled two soccer-robot team strategies using POS and have demonstrated how it is a suited theoretical tool and also leads to extremely compact and modular code. It turns out these two strategies perform very well in a JavaSoccer tournament against other predefined strategies.

7. Conclusion

7.1. Summary

This article establishes a theoretical basis for studying interaction protocols in multi-agent systems. Such a notion has been clearly defined as well as related concepts that sometimes stay vague in the literature. We have set up the basic elements for an interaction protocols' semantics in a similar way as what has been carried out for programming languages. We have shown on an extended example how this formalism is effective. The foremost advantage of POS is that it provides a large power of expression to interaction protocols (actually a maximal power, i.e., POS can emulate Turing complete calculus systems). The analogy made with programming languages enabled to look at the concept of interaction protocols in a new fashion. Besides, POS provides a well defined formal framework.

At the end of section 5.2 we introduced a first constructor/operator on protocols. We are currently working on the notion of parameterized protocol. In fact, the issue here is to manage a set of production rules (SianN) that is like a program whose argument is n (i.e., the number of agents participating in the conversation). One thus has $\text{SianN}(n) = \text{SianN}[N = n]$. Therefore, when an agent decides to start communicating with this protocol, it "loads" $\text{SianN}(n)$.

The way our protocols have been written using POS greatly resembles functional programming. Arguments are passed from parameterized states to parameterized states as from functions to functions in functional programming, and a lot of rewriting is performed based on the pattern of objects one manages.

7.2. *Perspectives*

In the perspective of a validation and evaluation of protocols, we are identifying properties on protocols and conversations. Properties could be of a syntactic level (as often seen in the literature) but also of a semantics level. These latter properties deal with the protocols' side effects, i.e., the protocols' meaning. These side effects are restricted to what the agents using the protocols are capable of doing. They are not restricted in any other way. The advantage of this is that one can contemplate performing pretty much anything. The drawback is that this prevents from stating a lot of general semantic properties upon the protocols. Those properties will rather be specific to each protocol. However, let us recall that the same is true for programming languages where side effects are very simple and totally contained.

Thus POS has nothing to be ashamed of. As far as semantic properties, the work done for programming languages consisted in defining formalisms for proving programs such as Hoare's logic²⁸. It would be interesting to see whether it is possible to find an equivalent of Hoare's logic for interaction protocols, but this is out of this paper's scope.

Up to now, protocols are mainly built in one shot. POS allows to build protocols based on other protocols, e.g., by combining several protocols. In order to achieve this, we are currently defining operators on protocols that will allow to combine them. This notion of protocol algebra will allow to reach a higher order. It means leaving the door wide open to modular methods for the design of protocols and also to the automatic generation of protocols.

7.3. *Discussion*

AgentTalk⁹ brings into play concepts somewhat close to POS. It is a programming language capable of implementing protocols and agents that behave according to a protocol. Furthermore, AgentTalk is concerned with the interactions among agents; it is not concerned with a detailed implementation of agents. The main difference with POS lies in that it comes down to a programming language, which implies it is very close to an implementation level whereas POS is meant to be both a formal specification language and a programming language.

AgentTalk describes a protocol by means of finite state automata augmented with a set of variables that can be tested. Like POS one goes from one state to another upon sending or receiving a message which are syntactically simple (as are finite states). Actions to be performed are defined for each transition. The general idea is very similar to POS' rules which are called *scripts*. However, the implementations (i.e., scripts) are much less readable than POS' rules. All of the protocols used as

examples could as well be represented by simple finite state protocols. In practical terms, POS has thus more expression power.

As emphasized in the literature, AgenTalk's protocols are capable of defining interfaces of the side effects to be produced (which are called *agent functions*). And agents are free to implement them as they want to. This allows to make a clear distinction between the agents' protocols and their internal architecture. Thus, a common feature with POS is the possibility to define the protocols' semantics. Besides this can be achieved in a very similar way.

Another common feature with POS is that AgenTalk's protocols can be built in a highly modular way with an inheritance mechanism quite similar to object-oriented languages. But as far as POS, the door is left open to other possible operations for combining protocols.

Finally, AgenTalk²⁹ enables the handling of several conversations at the same time as in POS. But it is not a specification language and has a much lower expression power. It introduces meta-protocols described as simple protocols, in order to control the overall system. A similar concept has been developed for POS.

References

1. M. P. Singh. Toward interaction oriented programming. In *Second International Conference on Multi-Agent Systems (ICMAS-96)*, Tokyo, Japan, December 1996.
2. Mark d'Inverno, David Kinny, and Michael Luck. Interaction protocols in Agentis. In Yves Demazeau, editor, *Third International Conference on MultiAgent Systems (ICMAS-98)*, pages 112–119, Paris, France, July 1998. IEEE.
3. R.G. Smith and R. Davis. Framework for cooperation in distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):61–70, January 1981.
4. Sati Singh Sian. Adaptation based on cooperative learning in multi-agent systems. In Yves Demazeau and Jean-Pierre Muller, editors, *Decentralized AI*, volume II, pages 257–272, Amsterdam, The Netherlands, 1991. Elsevier Science Publishers B.V.
5. Tuomas Sandholm and Victor Lesser. Issues in automated negotiation and electronic commerce: Extending the Contract Net framework. In *First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 328–335, San Francisco, USA, June 1995. AAAI Press.
6. Birgit Burmeister, Afsaneh Haddadi, and Kurt Sundermeyer. Generic, configurable, cooperation protocols for multi-agent systems. In C. Castelfranchi and Jean-Pierre Muller, editors, *From Reaction to Cognition*, volume 957 of *Lecture notes in AI*, pages 157–171, Berlin, Germany, 1995. Springer Verlag. Appeared also in MAAMAW-93, Neuchatel.
7. David Kinny. The Agentis interaction model. In *Fifth International Workshop on Agent Theories, Architectures and Languages (ATAL-98)*, pages 247–260, Paris, France, July 1998.
8. Mihai Barbuceanu and Wai-Kao Lo. Conversation oriented programming in COOL: Current state and future directions. In *Autonomous Agents'99 Special Workshop on Conversation Policies*, 1999.
9. Kazubiro Kuwabara, Toru Ishida, and Nobuyasu Osato. AgenTalk: Describing multi-agent coordination protocols with inheritance. In *Seventh International Conference on Tools with Artificial Intelligence*, pages 460–465, Herndon, Virginia, November 1995.

10. R. S. Cost, Y. Chen, T. Finin, Y. Labrou, and Y. Peng. Modeling agent conversation with colored Petri nets. In Jeff Bradshaw, editor, *Autonomous Agents'99, Special Workshop on Conversation Policies*, May 1999.
11. J.-L. Koning, G. Francois, and Y. Demazeau. Formalization and pre-validation for interaction protocols in multiagent systems. In Henri Prade, editor, *13th European Conference on Artificial Intelligence (ECAI-98)*, pages 298–302, Brighton, UK, August 1998. John Wiley & Sons.
12. Jean-Luc Koning and Pierre-Yves Oudeyer. Modeling soccer-robots strategies through conversation policies. In F. Mattern and D. Kotz, editors, *2nd IEEE International Symposium on Agent Systems and Application (ASA/MA-00)*, Zurich, Switzerland, September 2000. Springer Verlag.
13. R. Milner. A proposal for standard ML. In *ACM Conference on Lisp and Functional Programming*, 1987.
14. Thomas A. Sudkamp. *Languages and Machines*. Addison-Wesley Publishing Company, Inc., 1988.
15. G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus university, Computer Science Department, Denmark, 1981.
16. F. S. de Boer et al. Formal semantics for an abstract agent programming language. In Munindar P. Singh, Anand S. Rao, and Michael J. Wooldridge, editors, *4th International Workshop on Agent Theories, Architectures, and Languages (ATAL-97)*, Providence, Rhode Island, 1997.
17. Angela Dalmonte and Mauro Gaspari. Modelling interaction in agent systems. In *14th International Joint Conference in Artificial Intelligence (IJCAI-95)*, Montreal, August 1995.
18. M. Hennessy. *The Semantics of Programming Languages: An Introduction Using Structured Operational Semantics*. Wiley, 1990. Out of print.
19. J. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, 1969.
20. F. C. Hennie. *Introduction to Computability*. Addison-Wesley, Reading, MA, 1977.
21. P. Hudak and P. Wadler. Report on the programming language Haskell. Technical Report YALE/DCS/RR-777, Yale University, 1990.
22. Dominique Bolignano. An approach to the formal verification of cryptographic protocols. In *3rd ACM Conference on Computer and Communications Security*, March 1996.
23. Martin Odersky and Philip Wadler. Pizza into Java: Translating theory into practice. In *4th ACM Symposium on Principles of Programming Languages*, Paris, France, January 1997.
24. A. Tanenbaum. *Operating Systems Design and Implementation*. Prentice Hall, Inc., 1987.
25. Jean-Luc Koning and Pierre-Yves Oudeyer. Modeling and implementing conversation policies using POS. In B. d'Auriol, editor, *International Conference on Communications in Computing (CIC-00)*, Las Vegas, NV, June 2000. CSREA Press.
26. R. Alami and S. Fleury. Multi-robot cooperation in the MARTHA project. *Robotics and Automation*, 5(1), March 1998.
27. Javabots information page. <http://www.cs.cmu.edu/trb/JavaBots>.
28. C. A. R. Hoare and P. E. Lauer. *Consistent and Complementary Formal Theories of the Semantics of Programming Languages*. -, 1974.
29. Kazubiro Kuwabara, Toru Ishida, and Nobuyasu Osato. AgenTalk: Coordination protocol description for multiagent systems. In *First International Conference on Multi-Agent Systems (ICMAS-95)*, San Francisco, June 1995. AAAI Press. Poster.